1. 上课约定须知

课程主题：SparkCore第三次课：Spark App 提交和 SparkContext 初始化
上课时间：20:00 - 23:00
课件休息：21:30 左右 休息10分钟
课前签到：如果能听见音乐，能看到画面，请在直播间扣 666 签到

2. 上次作业复盘

使用 Spark 的 RPC 框架实现一款 C/S 架构的聊天应用程序。

要求：

```
1、该应用应用程序，有最基本的服务端程序和客户端程序
2、首先启动服务端，保证服务端一直运行
3、然后启动客户端，客户端向服务端注册
4、再启动其他多个客户端，待启动的客户端注册之后，和其他客户端（当然，当前客户端必须具备感知其他客户端存在的功能，然后选择通信对象）进行通信。
5、客户端关闭，其他客户端收到通知。
```

3. 上次内容总结

上次课程的主要内容是：源码分析1：Spark RPC 和 集群启动源码分析

```
1、Spark RPC
    1.x 版本的基于 Akka 实现
        模拟实现 YARN（资源调度系统）
    2.x 版本的基于 Netty 实现
        模拟了一次网络通信 + 布置作业

2、集群启动脚本脚本 start-all.sh 分析
    Master启动：spark-class org.apache.spark.deploy.master.Master
    Worker启动：spark-class org.apache.sprak.deploy.worker.Worker

3、Spark Stanalone 集群启动：Master 启动分析
    1、启动 rpc 服务端
    2、启动 web ui
    3、选举 active master
    4、启动了一个定时任务：每隔一段时间去检查 dead workers

4、Spark Stanalone 集群启动：Worker 启动分析
    1、启动 RPC 服务
    2、启动 web ui
    3、先向 Master 注册
    4、注册成功之后，定时发心跳
```

4. 本次内容大纲

今天讲解的主要内容是：Spark Application 提交 和 SparkContext 的初始化源码分析

```
1、Spark Application 应用程序编写套路总结，找到程序执行入口

2、spark-submit 脚本分析

3、SparkSubmit 类分析

4、Client提交Job到Master过程解析

5、Driver启动和SparkContext初始化源码详解
    SparkConf 初始化
    SparkEnv 初始化
    DADScheduler，TaskScheduler，SchedulerBackEnd 初始化
    SchedulerBackEnd启动

6、Applicatoion注册

7、Executor启动分析

8、RDD的DAG构建和Stage切分源码详解
```

```
9、TaskScheduler提交Task分析
```

这是今天的内容，下一次课的内容就是 Task 的执行！

5. 详细课堂内容

5.1. Spark 程序编写套路总结

在 Spark 的源码项目中，其实有一个 JavaWordCount 的入门程序！

Spark 程序的编写套路：标准五步走

```
1、重点：获取程序编写入口SparkContext
    new SparkContext(sparkConf)
2、通过SparkContext来加载数据源得到数据抽象对象：RDD
3、针对数据抽象对象RDD调用各种算子执行各种逻辑计算：lazy，延迟到action的内部来执行
4、重点：调用action算子触发任务的提交执行
    sparkContext.runJob()
5、处理结果并且关闭资源
```

去源码项目中找到示例程序，分析程序，得到程序执行入口：

```
// 初始化执行环境
1、SparkSession spark =
SparkSession.builder().appName("JavaWordCount").getOrCreate();

// 触发 job 的提交
2、counts.collect();
```

关于 Application 在执行过程中的几对概念：

```
1、Master + Worker
2、Driver + Executor
3、Application + Job + Stage + Task
```

5.2. spark-submit 脚本分析

写好一个程序，打成 jar 包，然后通过 spark-submit 提交：http://spark.apache.org/docs/latest/submitting-applications.html

```
提交程序的命令：
./bin/spark-submit \
  --class org.apache.spark.examples.SparkPi \
  --master spark://207.184.161.138:7077 \
  --deploy-mode cluster \
  --supervise \
  --executor-memory 20G \
  --total-executor-cores 100 \
  /path/to/examples.jar \
  1000
```

```
spark-class org.apache.spark.deploy.SparkSubmit $CLASS
spark-class org.apache.spark.deploy.SparkSubmit SparkPi
```

```
java org.apache.spark.deploy.SparkSubmit args
```

最终跳转到 SparkSubmit 的 main() 方法

5.3. SparkSubmit 分析

核心入口：

```
SparkSubmit.main();
    val submit = new SparkSubmit();
    submit.doSubmit(args);
        SparkSubmit.submit(appArgs, uninitLog);
            runMain(args, uninitLog);
                // 根据 spark-submit 脚本的运行参数来决定，怎么去初始化 app
                val (childArgs, childClasspath, sparkConf, childMainClass) =
prepareSubmitEnvironment(args);
                // 调用 app.start() 启动提交
                app.start(childArgs.toArray, sparkConf)
```

关于 app 启动类：

```
// TODO_MA 注释： YARN 集群运行主类
private[deploy] val YARN_CLUSTER_SUBMIT_CLASS =
"org.apache.spark.deploy.yarn.YarnClusterApplication"
// TODO_MA 注释： Rest Cluster 运行主类
private[deploy] val REST_CLUSTER_SUBMIT_CLASS =
classOf[RestSubmissionClientApp].getName()
// TODO_MA 注释： StandAlone 运行主类
private[deploy] val STANDALONE_CLUSTER_SUBMIT_CLASS =
classOf[ClientApp].getName()
// TODO_MA 注释： Kubernetes 运行主类
private[deploy] val KUBERNETES_CLUSTER_SUBMIT_CLASS =
"org.apache.spark.deploy.k8s.submit.KubernetesClientApplication"
```

最后的总结：基于 SparkSubmit 的分析，得知，一个 Spark Job 的提交过程中，到底是使用哪个客户端来提交 job 到就集群运行。

5.4. Client 提交 Job 到 Master 过程解析

记住，在封装 RequestSubmitDriver 消息的时候，指定了 Driver 的启动类为：org.apache.spark.deploy.worker.DriverWrapper

```
// 调用 app.start() 启动提交
app.start(childArgs.toArray, sparkConf)
    // 发送 RequestSubmitDriver 消息给 Master
    val rpcEnv = RpcEnv.create("driverClient", Utils.localHostName(), 0, conf,
new SecurityManager(conf))
        rpcEnv.setupEndpoint("client", new ClientEndpoint(rpcEnv, driverArgs,
masterEndpoints, conf))
            // 当上面的代码执行的时候，应该跳转到
            // 1、ClientEndpoint 的构造方法
            // 2、ClientEndpoint 的 onStart() 方法
            ClientEndpoint.onStart()
                // ClientApp 中的通信终端 ClientEndpoint 发送 RequestSubmitDriver
消息给 Master，Master返回 SubmitDriverResponse
                asyncSendToMasterAndForwardReply[SubmitDriverResponse]
(RequestSubmitDriver(driverDescription))
```

然后 Master 端执行 Driver 处理：

```
// 创建 Driver 对象
val driver = createDriver(description)

// 持久化 Driver
persistenceEngine.addDriver(driver)
waitingDrivers += driver
drivers.add(driver)

// 开始调度
schedule()

// 返回响应
context.reply(SubmitDriverResponse(self, true, Some(driver.id), s"Driver
successfully submitted as ${driver.id}"))
```

由 schedule() 方法的内部实现可知：

```
1、Driver 的启动入口是： lauchDriver(worker, driver)
2、Executor 的启动入口： startExecutorsOnWorks()
```

5.5. Driver 启动和 SparkContext 初始化源码详解

5.6. Spark Job 启动的完整流程简述

```
*  1、编写代码
*  2、打成jar包
*  3、通过 spark-submit 脚本来提交
*  4、执行 SparkSubmit 类的  main ()
```

```
*  5、在保准的 spark Standalone 集群中：  转交给 ClientApp 的类来执行
*  6、会初始化 CLientEndponit（存在于client中） 的组件：发送 RequestSubmitDriver 给 Master
*      跟 Driver 程序中的 ClientEndpoint 不是同一个
*  7、Master 处理这个消息，然后就注册了： Driver（后续调用 scheduler() 方法来启动 Driver 和 Executor ）
*  8、启动Driver： java DriverDrapper 这个类，转到： DriverDrapper main（） 方法
*  9、通过反射的方式启动和执行 我们自己写的业务代码的 main() 方法： JavaWordCount.main()
*  10、自己编写的业务代码中的第一句代码： 初始化SparkSessoin(SparkConf， SparkContext)
*  11、初始化 SparkContext： TaskScheduler SchedulerBackend（DriverEndpoint CLientEndpoint） DAGScheduler
*  12、应用注册：CLientEndpoint 发送 RegisterApplication 消息给 Master，返回 RegisteredApplication
*  13、Master 发送消息 LaunchExecutor 给： Worker，启动 Executor（真正启动的是：ExecutorBackend）
*  14、Executor 启动了，则初始化一个线程池，等待 Driver 分发任务过来，由线程池执行
        Executor 启动好了之后，会向 Driver 注册，同时也会向 Master 反馈！
```

到此为止，一个 Application 或者说是一个 job 最终执行的时候需要的 Driver 和 Executor 等通过这个流程就完成了启动！

5.6.1. Driver 启动

详细内容，请看 Master.schedule() 方法的具体实现！

核心入口是：

```
launchDriver(worker, driver)
```

内部具体实现是，首先 Master 发送 LaunchDriver 消息给 Worker，然后 Worker 接收到 LaunchDriver 之后，就开始执行处理：

```
val driver = new DriverRunner(conf, driverId, workDir, sparkHome,.....)
driver.start()
```

开始启动 Driver，注意 Driver 的启动类是： DrvierWrapper

5.6.2. SparkContext 初始化

核心入口：

```
new SparkContext(sparkConf)
```

在 SparkContext 初始化过程中需要做的事情比较多，大概有：

```
01、创建Spark执行环境SparkEnv;
02、创建并且初始化Spark UI;
03、hadoop相关配置以及Executor环境变量的设置;
04、创建心跳接收器 HeartbeatReceiver

// 后续这三步是最重要的！TaskScheduler SchedulerBackend DAGScheduler
// SchedulerBackend 和 DAGScheduler 是 TaskScheduler 的成员变量！
```

05、创建任务调度 TaskScheduler 和 SchedulerBackend;
06、创建和启动 DAGScheduler;
07、TaskScheduler 的启动;

08、初始化BlockManager（BlockManager是存储体系的主要组件之一）
09、启动测量系统MetricsSystem;
10、创建和启动 Executor 分配管理器 ExecutorAllocationManager;
11、ContextCleaner 的启动和创建。
12、调用 setupAndStartListenerBus() 启动 ListenerBus
13、Spark 环境更新
14、通过 postApplicationStart() 向 ListenerBus 提交 SparkListenerApplicationStart 消息
15、创建 DAGSchedulerSource 和 BlockManagerSource;
16、注册钩子防止 SparkContext 的 stop() 钩子

其中，第 5、6、7 三大步骤是最重要的！重点关注对象：前提对象： SparkConf + SparkEnv

1、TaskScheduler，实现类是：TaskSchdulerImpl

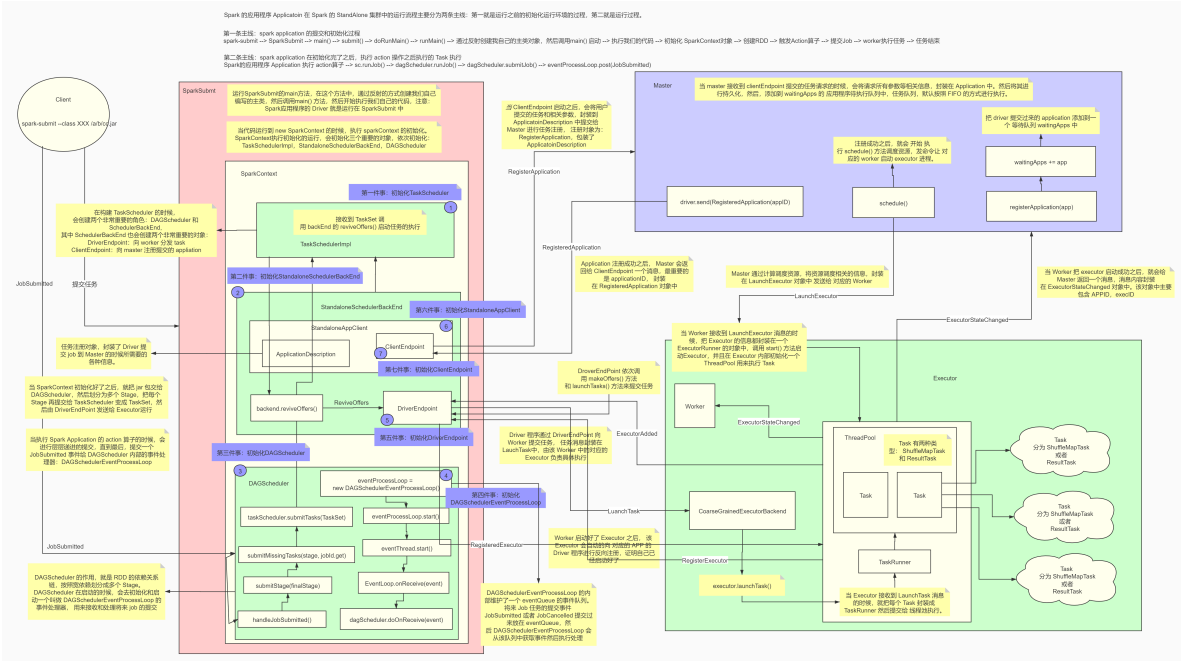
2、DAGScheduler
内部工作：DAGSchedulerEventProcessLoop 初始化和启动两件事

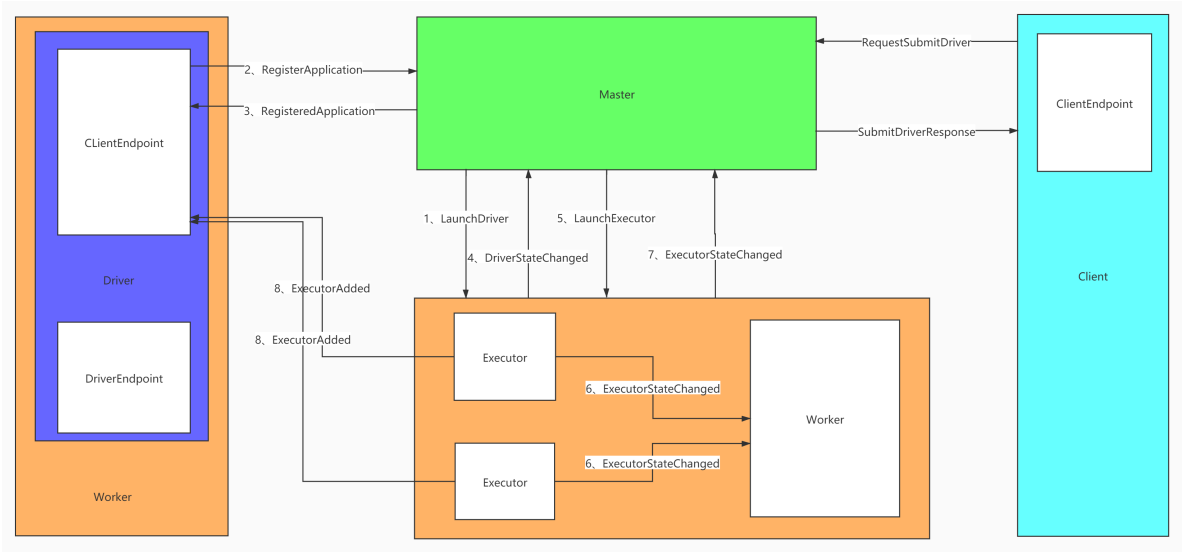
3、CoarseGrainedSchedulerBackend
backend.start()
有两个成员变量：
1、负责和 worker 通信： DriverEndpoint：进行 Task 的派发
2、负责和 master 通信： StandaloneAppClient 内部封装了 ClientEndpoint：进行 Application 的注册

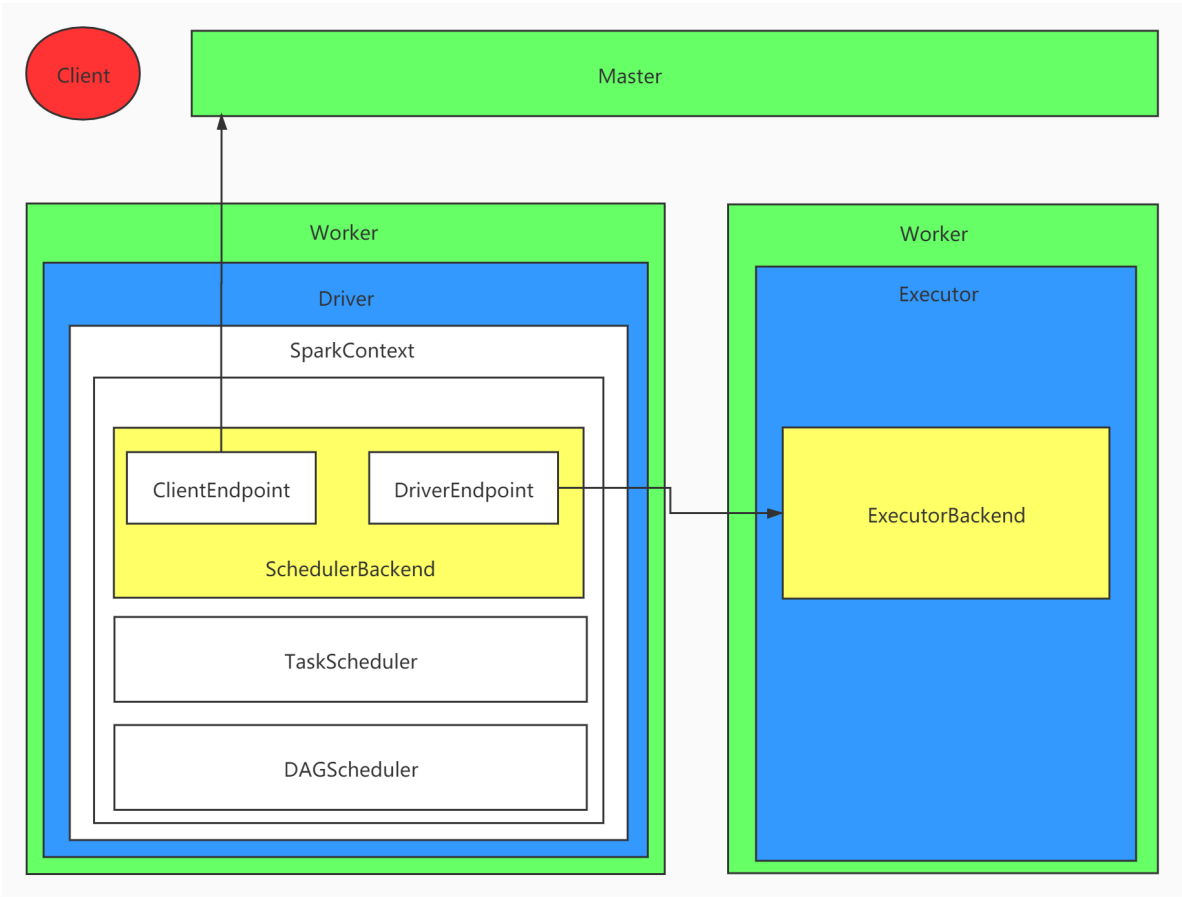
4、CoarseGrainedExecutorBackend

Driver + Executor（就是在 CoarseGrainedExecutorBackend 里面启动的）

详细流程，见图，见源码注释：







完整的总结：

```
SchdulerBackEnd  存在于Driver端
ExecutorBackEnd  存在于Executor端
如果Driver和Executor 通信，最终的处理组件就是： BackEnd
```

getPrefferedLocs() 获取数据本地性

5.7. SparkEnv 初始化

SparkContext 是一个非常重量化的组件：在一个JVM中最多只能存在一个，如果你想要新创建一个，则必须关闭之前的哪一个。

```
SparkContext
    SparkConf
    SparkEnv
        SecurityManager
        SerializerManager
        NettyBlockTransferService
        OutputCommitCoordinator
        .....
```

关于在 SparkContext 初始化过程中，初始化了 SparkEnv 对象，里面包含的主要组件和初始化步骤，如下：

```
01、创建安全管理器 SecurityManager
02、创建 RPC 通信层 RpcEnv
03、创建序列化管理器 SerializerManager
04、创建广播管理器 BroadcastManager
05、创建 Map 任务输出跟踪器 MapOutputTracker
06、创建 ShuffleManager
07、创建内存管理器 MemoryManager
08、创建块传输服务 NettyBlockTransferService
09、创建 BlockManagerMaster
10、创建块管理器 BlockManager
11、创建测量系统 MetricsSystem
12、创建 OutputCommitCoordinator
13、创建 SparkEnv
14、返回 SparkEnv
```

5.8. Applicatoion 注册

核心流程在：

```
StandaloneSchedulerBackend.start()
    // 启动DriverEndpint，负责跟Executor 打交道额
    super.start()

    // 注册对象
    val appDesc = ApplicationDescription(sc.appName,.....)

    // 启动了 CLientEndpoint，负责跟Master打交道： 进行job 的注册
    client = new StandaloneAppClient(sc.env.rpcEnv, masters, appDesc, this, conf
    client.start()
        endpoint.set(rpcEnv.setupEndpoint("AppClient", new
ClientEndpoint(rpcEnv)))
            ClientEndpoint.onStart()
                // 注册流程
                registerWithMaster(1)
```

注册中，Driver 会向 Master 发送 RegisterApplication 消息进行注册，然后接收到 Master 的 RegisteredApplication 消息，表示注册成功！则此时，Master 调度 schedule() 进行 Executor 的启动！

核心方法是：

```
startExecutorsOnWorkers()
```

5.9. Executor 启动分析

注意：Executor 的启动类是： org.apache.spark.executor.CoarseGrainedExecutorBackend

所以核心代码入口：

```
CoarseGrainedExecutorBackend.main()
```

在 Executor 启动中，最重要的事情：

```
1、向 Driver 进行注册，发送 RegisterExecutor 消息给 Driver
2、在接收到 Drvier 返回的 RegisteredExecutor 消息就构造一个 Executor 来启动，在这其中，
最重要的事情，是初始化一个线程池！
```

5.10. RDD 的 DAG 构建和 Stage 切分源码详解

Job 的提交入口是：应用程序中的 action 算子，比如 JavaWordCount 中：

```
List<Tuple2<String, Integer>> output = counts.collect();
```

内部调用 SparkContext 来提交！

```
sc.runJob(this, .....)
```

内部调用：

```
dagScheduler.runJob(rdd, cleanedFunc, partitions, callSite, resultHandler,
localProperties.get)
```

内部实现：

```
eventProcessLoop.post(JobSubmitted(jobId, rdd, func2, partitions.toArray,
callSite, waiter, SerializationUtils.clone(properties)))
```

跳转到：

```
DAGSchedulerEventProcessLoop.onReceive(event: DAGSchedulerEvent)
    dagScheduler.handleJobSubmitted(jobId, rdd, func, partitions, callSite,
listener, properties)
        // stage 切分
        finalStage = createResultStage(finalRDD, func, partitions, jobId,
callSite)
        // 提交 stage 执行
        submitStage(finalStage)
```

核心入口：

```
DagScheduler.submitStage(finalStage);
    submitMissingTasks(stage, jobId.get)
         taskScheduler.submitTasks(new TaskSet(tasks.toArray, stage.id,
stage.latestInfo.attemptNumber, jobId, properties))
```

到此，DAGScheduler 的工作完成。

5.11. TaskScheduler 提交 Task 分析

按照上述描述：TaskScheduler 提交 Task 的具体逻辑的入口：

```
taskScheduler.submitTasks(new TaskSet(tasks.toArray, stage.id,
stage.latestInfo.attemptNumber, jobId, properties))
```

内部调用：

```
backend.reviveOffers()
```

开始提交 Task：

```
launchTasks(taskDescs)
    executorData.executorEndpoint.send(LaunchTask(new
SerializableBuffer(serializedTask)))
```

发送 LaunchTask 消息给 Executor。当 CoarseGrainedExecutorBackend 接收到 LaunchTask 消息，则开始执行调用对应的 Executor 来执行 Task：

```
executor.launchTask(this, taskDesc)
```

具体的内部实现是：

```
val tr = new TaskRunner(context, taskDescription)
runningTasks.put(taskDescription.taskId, tr)
threadPool.execute(tr)
```

此刻，终于把 Task 提交到 Worker 上的 Executor 中的线程池来进行运行了。！

6. 本次课程总结

今天的内容，主要讲解，在 Spark Standalone 模式下，一个 Application 是怎么被提交到 集群 来运行的，以及为了运行，集群会怎么运行和调度呢？涉及到的主要内容有：

```
1、Spark Application 应用程序编写套路总结，找到程序执行入口

2、spark-submit 脚本分析
```

```
3、SparkSubmit 类分析

4、Client 提交 Job 到 Master 过程解析

5、Driver 启动和 SparkContext 初始化源码详解
    SparkConf 初始化
    SparkEnv 初始化
    DADScheduler，TaskScheduler，SchedulerBackEnd(DriverEndpoint,ClinetEnpoint)
初始化
    SchedulerBackEnd 启动

6、Applicatoion 注册

7、Executor 启动分析
```

还有两个知识没有讲完，留到下节课：

```
8、RDD 的 DAG 构建和 Stage 切分源码详解

9、TaskScheduler 提交 Task 分析
```

7. 本次课程作业

使用 Spark 的 RPC 框架实现一款 C/S 架构的聊天应用程序。

要求：

```
1、该应用应用程序，有最基本的服务端程序和客户端程序
2、首先启动服务端，保证服务端一直运行
3、然后启动客户端，客户端向服务端注册
4、再启动其他多个客户端，待启动的客户端注册之后，和其他客户端（当然，当前客户端必须具备感知其他
客户端存在的功能，然后选择通信对象）进行通信。
5、客户端关闭，其他客户端收到通知。
```